Sun, W., Mouskos, C. K., and Bernstein, D.

A Web-Based Parking Reservation System

Wu Sun
New Jersey Transportation Information and Decision Engineering (TIDE) Center
New Jersey Institute of Technology
Univeristy Heights
Newark, NJ 07102
Tel. (973)5965705
Fax.(973)5965206
Email: wu.sun@njit.edu

Kyriacos C. Mouskos,
Research Professor, Institute for Transportation Systems
The City College of New York
Convent Ave., Bldg. Y-220
NY, NY 10031
Tel. (212)6508047
Fax. (212)6508374
Email: mouskos@ti-mail.engr.ccny.cuny.edu

and

David Bernstein
Department of Computer Science
James Madison University
Harrisonburg, VA 22801
Tel. (540)5681671
Email: bernstdh@jmu.edu

Word Count: 6233 (text 4483, figures 1750)

ABSTRACT

The operation of both terminal parking facilities and park-and-ride facilities can be improved by a parking reservation system. The purpose of this paper is to describe one such system. The Web-PRS is a Java-based system using Extensible Markup Language (XML) files for back-end data storage. The model solver is a C language based ILOG CPLEX optimizer, which is integrated into Web-PRS using JNI (Java Native Interfaces)-Based ILOG Concert Technology.

INTRODUCTION

The purpose of this paper is to discuss the development of an online interactive parking reservation system. Our goal in developing this system is to improve the operation of both terminal parking facilities and park-and-ride facilities.

Our interest in improving terminal parking facilities is motivated by the observation that a major cause of congestion and wasted travel time, especially in business districts, is parking search. That is, people often "circle around" looking for a good parking space. This causes congestion both because it adds to the number of vehicles on the road at any point in time and because people tend to drive fairly slowly when searching. Our interest in improving park-and-ride facilities is motivated by the observation that some of the reasons that people give for not using such facilities include the inconvenience, method of payment, and increase in travel time.

A recent paper by Mouskos, Bernstein and Tavantazis *(1)* describes a Parking Reservation Problem that can be used to eliminate or reduce these problems with parking facilities. Specifically, their paper describes both a deterministic and a stochastic model that can be used to make parking reservations. Both models can be solved with commercially available linear programming solver. In fact, reasonably large problems have been solved using the commercial solver CPLEX.

This paper discusses how a complete WWW-based system (that makes use of these models) can be implemented. After obtaining information from the user, the system first solves a parking reservation problem (with one of two objectives -- revenue maximization or user cost minimization). Next, a shortest path problem is solved on a network composed of parking lots, destination office buildings, and connecting paths. The shortest path result represents the shortest walking path from a user's assigned parking lot to his/her office building.

We begin by discussing the parking reservation problem. Next, we discuss the commercial solver that is used to solve this problem. After that, we discuss the design of the overall system. Finally we describe the data structures/models that are used.

MODEL FORMULATION

The deterministic parking reservation problem is formulated as *(1):*

Let:

$$x_{ijk} = \begin{cases} 1 & \text{if } i\text{th car parks in parking lot } j \text{ at period } k \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ijk} = \begin{cases} 1 & \text{if } i\text{th car that is parked in lot } j \text{ leaves at period } k \\ 0 & \text{otherwise} \end{cases}$$

Z_j is the capacity of parking lot j,

c_{ijk} is the cost of vehicle i to park in parking lot j at period k,

where:

i = 1,……., I; I is the total number of cars,
j = 1,……, J; J is the total number of parking lots,
k = 1,……, K; K is the total number of time periods,

The objective of this system is to minimize total system wide parking cost, we assume the costs for parking include a fixed monetary value for each parking lot and a fixed travel cost (walking time) from the parking lot to the final destination.

$$\min \sum_{ijk} c_{ijk} x_{ijk} \qquad (1)$$

subject to a number of constraints. The assignment constraints require that each vehicle i is assigned to parking lot j at period k, therefore one of the x_{ijk}'s becomes one and the remaining are assigned zero values. This can be represented as follows:

$$\sum_{j} x_{ijk} = 1 \quad \text{for all } i,k \qquad (2)$$

The parking lot capacity constraints ensure that the number of vehicles assigned to parking lot j at period k should be less than or equal to the number of available parking spaces in lot j. For the first time period 1 (k=1) we assume there are no vehicles parked, hence the constraint can be represented as:

$$\sum_{i} x_{ij1} \le z_j \qquad (3)$$

For time period k, the number of new vehicles that can be assigned to parking lot j must be less or equal to the capacity of the lot minus the number of spaces occupied plus the number of spaces available due to vehicles that have left at period k. Hence, these constraints can be represented as follows

$$\sum_{i} x_{ijk} \le z_j - \sum_{i} \sum_{k'=1}^{k-1} x_{ijk'} + \sum_{i} \sum_{k'=1}^{k} y_{ijk'} \qquad (4)$$

$$x_{ijk} = y_{ijl} \qquad \text{where } l > k \qquad (5)$$

Assumptions:

1. The time period that vehicle *i* arrives and departs is known a priori.
2. The parking capacity is fixed and known.
3. A vehicle must arrive and depart at different time periods.
4. We assume that there is sufficient parking capacity to accommodate all incoming vehicles at any time period. For example one of the parking lots has sufficient capacity to accommodate all the vehicles at any time period, absorbing all the vehicles that are not assigned to the other parking lots.
5. We assume the costs for parking include a fixed monetary value for each parking lot and a fixed travel cost (walking time) from the parking lot to the final destination that is user specific.

Constraint (5) ensures that an vehicle i that is assigned to parking lot j at period k, is the same as the vehicle i that leaves from the same parking lot j at a later time period l that is pre-specified by vehicle i. Because of condition (5) the variables y_i are substituted in (4) and are no longer needed in the formulation. The above formulation has been shown to produce binary integer solutions by showing that the constraint matrix is totally unimodular *(1).*

MODEL SOLVER

The linear programming solver ILOG CPLEX is used to solve the deterministic model. The ILOG CPLEX delivers high-performance, robust, flexible optimizers for solving linear, mixed-integer and quadratic programming problems in mission-critical resource allocation applications *(6)*.

ILOG CPLEX Suite consists of the following components:

- CPLEX Simplex Optimizers
- CPLEX Mixed Integer Optimizer
- CPLEX Barrier Optimizer

CPLEX accepts the following ways of representing a mathematical programming problem:

- A text file, using CPLEX LP Format or industry standard MPS format
- ILOG Concert Technology, using modeling objects and algebraic expressions in C++ or Java
- Sparse Matrix representation, using matrix indices

The Web-PRS uses CPLEX Simplex Optimizer as its solver, and its mathematical programming model is represented as a CPLEXModeler object, a Java-Based ILOG Concert Technology modeling object.

The ILOG Concert Technology for the Java platform is a class library offering an API with modeling facilities you can use to embed CPLEX optimizers in a java application. The Concert Technology classes for CPLEX are implemented using Java Native Interfaces (JNI). On Windows system, this library is called cplex75.dll *(6)*.

To use the CPLEX Java interfaces, ilog.concert.* and ilog.cplex.* packages need to be imported into the application. The first task is to create an IloCplex object. The Interface functions for doing so are defined by the Concert Technology interface IloModeler and its extension IloMPModeler. In the CPLEXModeler object of Web-PRS, methods objGenerator(), assignmentConstraintsGenerator(), and capacityConstraintsGenerator() are designed for constructing objective function, assignment constraints and capacity constraints, respectively. After a model has been created, the IloCplex object is ready to solve the model. Invoking the optimizer is as simple as calling method solve(). Method solve() returns a Boolean indicating whether the optimization succeeded in finding a solution. More precise information about the outcome can be obtained by calling: IlogCplex.getStatus(). The objective value of that solution can be queried using method:Double lbjval=cplex.getObjValue(). Solution values for all the decision variables can be queried by calling:Double [] xval=cplex.getValues().

SYSTEM DESIGN

The Web-PRS was designed using UML, a modeling language for OO analysis and design. By highlighting important details in a concise and clear fashion, UML helps to achieve better communication and OO analysis *(2)*. The major UML techniques used in the Web-PRS are use cases, class diagrams, sequence diagrams, and state diagrams. To limit the length of this paper, use cases are omitted in this paper.

The Web-PRS is designed for two types of users, subscribers and non-subscribers. Subscribers are registered users, their information are stored in configuration XML files. Non-subscribers are users not registered with the Web-PRS. Non-subscriber information, including parking patterns, is collected on a one-time base for temporary use. The major functionality of Web-PRS is to provide subscribers and non-subscribers an interactive tool of making parking reservations online. The user specifies the time period of arrival and departure, and the system

assigns a parking space to him/her either directly on the screen or through a notification to their e-mail account or both. Furthermore, the system produces the shortest walking path from the assigned parking lot to the user's desired destination.

Architecture

The Web-PRS is a multi-tier system, including front end JSPs (JavaServer Pages), Servlet controller, CPLEX handler, and XML(Extensible Markup Languages) files. The front-end JSPs provide a group of interfaces to collect user entered information. The Servlet controller is used to control page transitions and pass data between front end and CPLEX handler and XML files. The CPLEX handler is a wrapper of an ILOG Concert Technology-Based Java object used to represent and solve the parking reservation model. The system configurations files, written as XML files, store subscriber information, parking facility information and model parameters of the PRS. Using XML files to store system configuration files increases the portability of the Web-PRS.

Class Diagrams

A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them. There are two principal kinds of static relationships, subtype and association. Associations represent relationships between instances of classes. Class diagrams also show the attributes and operations of a class and the constraints that apply to the way objects are connected *(2).* Web-PRS consists of three major classes:FrontEndController, PRSMetaDataHome, and CPLEXHandler.

FrontEndController:

A FrontEndController object contains a vector of SubscriberRequest objects, a vector of NonSubscriberRequest objects, a vector of Login objects, a PRSMetaDataHome object, a UserData object, and a PRSImageResult object. The PRSMetaHome object is instantiated during FrontEndController initialization. When FrontEndController detects that a user request is complete, it instantiates a CPLEXHandler object, which will be used to represent and solve the parking reservation model. The parking assignment result is delivered to the user in three formats, online text result, online graphic result, and result via Email. The graphic result is a graphics interchange format (GIF) file stored on the server's hard disk. The FrontEndController class diagram is presented in Figure 1.

A SubscriberRequest object contains an individual subscriber's request, such as name, arrival time, departure time, office addresses etc.. A NonSubscriberRequest contains similar information for non-subscribers. A UserData object is a collection of information of users already in the study parking facilities, including user ID assigned when a user request comes in, user in time, and user out time. For the current Web-PRS, it is assumed that in addition to users come in later, there are already a fixed number of users in the parking facility. The in times and out times of these users are randomly generated.

A PRSImageResult object handles the generation of a GIF file. It blends two images together, one image is a map of the study site, and the other image is a path representing the shortest walking distance from the assigned parking lot to users' office building. The study site map is loaded in the application, and converted to an off-screen Image object. The shortest path image is created in the application using network configuration and shortest path information.

CPLEXHandler:

A CPLEXHandler is instantiated in FrontEndController. In a CPLEXHandler object three main Java objects are instantiated, they are CPLEXModeler, SPHandler, and EmailHandler. CPLEXHandler class diagram is presented in Figure 2.

A CPLEXModeler object contains methods to construct a mathematical model in a format that can be solved using ILOG CPLEX. EmailHandler object handles sending parking assignments as an email message to users. The Web-PRS runs Dijkastra's shortest path algorithm to find the shortest path from assigned parking lot to user's destination. In Web-PRS, a Dijkastra object is used to implement the shortest path algorithm. The SPHandler object is a wrapper of the Dijkastra object. Each office building or parking lot is represented as a node, and is implemented as a Java object NetNode. CheckedNodeSet and NotCheckedNodeSet are two node sets composed of a vector of NetNode objects, representing checked nodes and non-checked nodes respectively.

PRSMetaDataHome:

A PRSMetaDataHome object contains Web-PRS system configuration data. A PRSMetaDataHome object has several Java object data members, including a vector of MetaDropDownMenu objects, a vector of MetaParkingLot objects, a MetaParameters object, a MetaNetwork object, and a vector of MetaSubscriber objects. A MetaParameters object includes a MetaDistribution object. A MetaNetwork object is composed of a number of MetaArc objects. A MetDropDownMenu object contains the content of a drop down menu on JSPs. Using MetaDropDownMenu objects on a JSP can avoid hard-coding of item names in a menu, and enforce unity of menus used on different JSPs. PRSMetaDataHome class diagram is presented in Figure 3.

Sequence Diagram

Interaction diagrams are models that describe how groups of objects collaborate in some behavior *(2)*. Sequence diagrams are one of two major types of interaction diagrams.

The FrontEndController coverts user entered information to SubscriberRequest and NonSubscriberRequest objects, and then pass them, together with system configuration information stored in PRSMetaDataHome object, to CPLEXHandler. SubscriberRequest and NonSubscriberRequest objects contain user requests collected from front-end JSPs. A sequence diagram is presented in Figure 4.

State Diagram

State diagrams describe all of the possible states that a particular object can get into and how the object's state changes as a result of events that reach the object. State diagrams are drawn for a single class to show the lifetime behavior of a single object *(2)*.

The FrontEndController is loaded (instantiated and have its init() called) on the startup of the Web-PRS, using a load-on-startup tag in the web.xml configuration file of Web-PRS. The content of this tag is assigned a value 0, indicting FrontEndController has the top loading priority. The JSP/Servlet container must guarantee that the servlets marked with lower integers are loaded before servlets marked with higher integers *(3)*.

A servlet is only loaded once initially and then services requests in multiple servlet container threads *(4)*. The initialization of a servlet is done in the init() method of the servlet. Before requests are serviced, it is recommended that certain operations, such as loading

persistent data, need to be included in servlet initialization. The initialization of the FrontEndController includes loading the Web-PRS configuration XML file and instantiating a PRSMetaDataHome object.

Each time a JSP request comes in, the name of the JSP page is passed in to FrontEndController, and an action takes place accordingly. The action could be simply a transition from one page to another page, or, it could be a more complicated process of generating either a SubscriberRequest object, a NonSubscriberRequest object, or a Login object using information wrapped in the JSP request. SubscriberRequest and NonSubscriberRequest are exclusive to each other. In no circumstance, for the same user, these two exist at the same time. The major function performed in the Login object is to validate whether a user name and password pair match. For a Web-PRS user, parking reservation results are generated in a CPLEXHandler object, and can only be generated after both a Login object and a SubscriberRequest or a NonSubscriberRequest object have already existed. Since the FrondEndController is multithreaded and JSP requests from different users may intermix, session ID of each JSP request is used as the key in matching user objects, namely Login, SubscriberRequest and NonSubscriberRequest objects. These three types of objects are stored in three corresponding vectors, which are checked by the readyToCreateResult() method to see if parking reservation results should be generated for the user represented by a given session ID. The state diagram of the FrontEndController is presented in Figure 5.

MISCELLANEOUS ISSUES

Image Blending

To generate a graphic parking reservation result, two images need to be blended together. The first image, a study site map is loaded and converted to an off-screen Image object in the application. The second image, a shortest path off-screen image, is created in the application. A straight-forward way of creating an Image object is using a java.awt.image.MemoryImageSource object and creating an Image object for it with java.awt.Toolkit's createImage(ImageProducer) method *(4)*. However, in order to take advantage of the high-level AWT (Abstract Windowing Toolkit) graphic methods, a Frame object is created and then connected to AWT toolkit by calling its addNotify() method *(4)*. This Frame object can be used to obtain an off-screen image with full AWT support.

The Java 2D API provides support for the blending of multiple images through what are known as Porter-Duff rules *(8)*. The rules describe how to combine the contents of multiple images when one image is drawn on top of the other. Within the Java 2D API, the blending rules are supported by the AlphaComposite class. The class provides twelve constants, one for each rule *(9)*.

Finally, the blended off-screen Image object is encoded as a binary stream in GIF format, which is understood by the browser. A free encoder package, Acme.JPM.Encoders.GifEncoder *(7)*, is available on the Internet. Using this package, it takes only a few lines of code to save an Image object to the hard drive as a GIF file.

Send Parking Reservation Results via Email

In order to provide email functionality programmatically, we need to use JavaMail APIs. A transport is Sun's term for a service that has the capability to send messages to its destinations, and a store is a service to retrieve messages delivered to a mailbox through other user's

transports *(4)*. The widely used transport protocol is the Simple Mail Transfer Protocol (SMTP), and the widely used store protocols are Post Office Protocol 3 (POP3) and Internet Message Access Protocol (IMAP). JavaMail APIs provide communication using these protocols. It is necessary to include both JavaMail API implementation and JavaBean Activation Framework (JAF) in the Web-Inf/lib directory, because JavaMail Messages depend on JAF. In the test example of Web-PRS, SMTP mail host of Web-PRS is set to mailhost.njit.edu.

SYSTEM DATA STRUCTURE

There are five types of configuration data in Web-PRS, the subscriber data, drop down menu data, parking lot data, network data, and system parameters. Each registered subscriber has a profile in the system, containing information from the subscriber's arrival and departure patterns to other personal information such as user name, password, address, etc.. Drop down menu data contains the contents of drop down menus on JSPs. Parking lot data describes the capacity, cost, and other information of all the study parking lots in the system. The configurations of the end network are described by the network data. System parameters critical to a Web-PRS include the type of the parking reservation model on which the Web-PRS is based, the type of shortest path algorithm which is used to calculate the path from the assigned parking lot to the destination, and the statistical distribution types of user arrival and departure times when the parking reservation model is stochastic.

Due to its hierarchical, easily processed, easily read, and stylable nature, XML is quickly becoming the standard of data interchange on the Internet. A configuration XML file, named as Parking Reservation Markup Language, is used in the Web-PRS to store all the above-mentioned five data types.

Parking Reservation Markup Language (PRML)

PRML, an XML application, is designed to specify the data format for exchange of parking reservation system information over the Internet. A sample PRML is presented in the next section with element descriptions attached. For simplicity reasons, the inner most elements are not closed. Elements enclosed in brackets are optional repetitive elements.

<webprsconfig>	root element
<subscribers>	encloses all subscribers of Web-PRS
<subscriber>	encloses one subscriber
<name>	subscriber name
<password>	password used to access Web-PRS
<email>	subscriber Email
<tel>	subscriber telephone number
<homeaddress>	subscriber home address
<officeaddress>	subscriber office address
<arrivaltime>	subscriber arrival time
<departuretime>	subscriber departure time
<age>	subscriber age
<gender>	subscriber gender
<income>	subscriber annual income
<costtopay>	parking cost subscriber willing to pay
</subscriber>	
[<subscriber>..	

```
        ...
        <subscriber>..
       ]
</subscribers>
<menus>                          encloses all drop down menus
       <menu>                    encloses one drop down menu
              <name>             drop down menu name
              <noofitems>        number of items in the menu
              <item>             encloses one item in the menu
              [<item>..
               ...
               <item>..
              ]
       </menu>
       [<menu>..
        ...
        <menu>..
       ]
</menus>
<lots>                                  encloses all parking lots
       <lot>                            encloses one parking lot
              <number>                  parking lot number
              <capacity>                parking lot capacity
              <cost>                    parking cost in the lot
       </lot>
       [
        <lot>..
        ...
        <lot>..
       ]
</lots>
<network>                               encloses a parking network
       <nodes>                          encloses all nodes in the network
              <node>                    encloses a node in the network
                     <number>           node number
                     <x>                x coordinate of this node
                     <y>                y coordinate of this node
              </node>
              [
              <node>..
              ...
              <node>..
              ]
       </nodes>
       <arcs>                           encloses all arcs in the network
              <arc>                     encloses a arc in the network
```

```
                        <staringnode>       arc starting node
                        <endingnode>        arc ending node
                        <length>            arc length
                    </arc>
                    [
                     <arc>..
                     ...
                    <arc>..
                    ]
                </arcs>
        </network>
        <modelparameters>                   encloses Web-PRS parameters
                <type>                      parking reservation system type
                <distribution>              arrival and departure distributions
                        <arrival>           arrival distribution
                        <departure>         departure distribution
                </distribution>
                <spalgorithm>               shortest path algorithm type
        </modelparameters>
</webprsconfig>
```

PRML Schema

The grammar of PRML is governed by an XML Schema, which is recommended by W3C (world wide web consortium) *(5).* The Schema of PRML is presented as following:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name= "webprsconfig" type= "webprsconfigtype"/>
<xs:comlexType name= "webprsconfigtype">
        <xs:all>
                <xs:element name= "subscribers" type= "subscriberstype"/>
                <xs:element name= "menus" type= "menustype"/>
                <xs:element name= "lots" type= "lotstype"/>
                <xs:element name= "network" type= "networktype"/>
                <xs:element name= "modelparameters" type= "modelparameterstype"/>
        </xs:all>
</xs:complexType>
<xs:comlexType name= "subscriberstype">
        <xs:sequence>
                <xs:element name= "subscriber" type= "subscribertype" maxOccurs=
"unbounded"/>
        </xs:sequence>
</xs:complexType>
<xs:comlexType name= "menustype">
        <xs:sequence>
                <xs:element name= "menu" type= "menutype" maxOccurs= "unbounded"/>
        </xs:sequence>
```

```
</xs:complexType>
<xs:comlexType name= “lotstype”>
        <xs:sequence>
                <xs:element name= “lot” type= “lottype” maxOccurs= “unbounded”/>
        </xs:sequence>
</xs:complexType>
<xs:comlexType name= “networktype”>
        <xs:sequence>
                <xs:element name= “node” type= “nodetype” maxOccurs= “unbounded”/>
                <xs:element name= “arc” type= “arctype” maxOccurs= “unbounded”/>
        </xs:sequence>
</xs:complexType>
<xs:comlexType name= “parameterstype”>
        <xs:all>
                <xs:element name= “type”>
                        <xs:simpleType>
                                <xs:restriction base= “xs:String”>
                                        <xs:enumeration value= “det”/>
                                        <xs:enumeration value= “sto”/>
                                        <xs:enumeration value= “dyn”/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:element>
                <xs:element name= “distribution” type= “distributiontype”/>
                <xs:element name= “spaalgorithm”>
                        <xs:simpleType>
                                <xs:restriction base= “xs:String”>
                                        <xs:enumeration value= “sp1”/>
                                        <xs:enumeration value= “sp2”/>
                                        <xs:enumeration value= “sp3”/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:element>
        </xs:all>
</xs:complexType>
<xs:comlexType name= “subscribertype”>
        <xs:all>
                <xs:element name= “name”>
                        <xs:simpleType>
                                <xs:restriction base= “xs:String”>
                                        <xs:minLength value= “5”/>
                                        <xs:maxLength value= “8”/>
                                        <xs:pattern value= “[a-zA-Z0-9]”/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:element>
```

```
        </xs:all>
</xs:complexType>
<xs:comlexType name= “menutype”>
        <xs:sequence>
                <xs:element name= “name” type= “xs:String”/>
                <xs:element name= “noofitems” type= “xs:integer”/>
                <xs:element name= “item” type= “xs:String” maxOccurs= “unbounded”/>
        </xs:sequence>
</xs:complexType>
<xs:comlexType name= “lottype”>
        <xs:sequence>
                <xs:element name= “number” type= “xs:String”/>
                <xs:element name= “capacity” type= “xs:integer”/>
                <xs:element name= “cost” type= “xs:decimal”/>
        </xs:sequence>
</xs:complexType>
<xs:comlexType name= “nodetype”>
        <xs:sequence>
                <xs:element name= “number” type= “xs:String”/>
                <xs:element name= “x” type= “xs:integer”/>
                <xs:element name= “y” type= “xs:integer” />
        </xs:sequence>
</xs:complexType>
<xs:comlexType name= “arctype”>
        <xs:sequence>
                <xs:element name= “startingnode” type= “xs:String”/>
                <xs:element name= “endingnode” type= “xs:String”/>
                <xs:element name= “length” type= “xs:decimal” />
        </xs:sequence>
</xs:complexType>
<xs:comlexType name= “distributiontype”>
        <xs:sequence>
                <xs:element name= “arrival” type= “arrivaldeparturetype”/>
                <xs:element name= “departure” type= “arrivaldeparturetype”/>
        </xs:sequence>
</xs:complexType>
<xs:simpleType name= “arrivaldeparturetype”>
        <xs:restriction base= “xs:String”>
                <xs:enumeration value= “norm”/>
                <xs:enumeration value= “other”/>
        </xs:restriction>
</xs:simpleType>
```

EXAMPLE

This system has been tested on parking facilities on the campus of New Jersey Institute of Technology (NJIT), Newark, New Jersey. The study parking lots include the parking deck and

parking lots 1, 2, 3, 5, 7, 12, and 16. All major intersections and building on the campus are represented as nodes in a network, and a Dijkstra's shortest path algorithm is run on the network. A subscriber logged in with his/her default profile information, and the Web-PRS produces both a text and a graphic result that includes: arrival time, departure time, assigned parking lot number, and shortest walking path and distance from assigned parking lot to his/her office building. The result is sent to the subscriber via both Internet and Email. This example contains a reasonably large problem with 500 randomly generated users in the parking lots, and an additional user added from Web-PRS user interfaces. The operation time of the parking lots are divided into 12 time periods. The text parking reservation result is presented in Figure 6, and the graphic result is presented in Figure 7.

CONCLUSIONS

Parking Reservation Systems aim to reduce the congestion associated with the search for parking and reduce the associated driver frustration, by providing the travelers the capability to reserve a parking space within the vicinity of their destination. Under this research, a prototype web-based interactive parking reservation system has been developed to assist travelers to reserve a parking space prior to their arrival at their destination through the Internet.

Two mathematical PRS formulations have been implemented that assume deterministic and stochastic arrivals and departures. Both mathematical formulations can be solved with commercially available linear programming solvers, such as CPLEX, which is the one used in this implementation. The present form of the Web-PRS is designed to handle both subscribers and non-subscribers. In this implementation only the deterministic mathematical formulation has been demonstrated. The ILOG Concert Technology is used to integrate Java-based interfaces and C language based CPLEX solver. The system has a multi-tier structure with configuration information stored in a back-end XML file. The use of a XML configuration file increases the portability of the Web-PRS. The Dijkstra's shortest path algorithm is integrated into the system to search for shortest walking path from assigned parking lot to the user's destination building. The Web-PRS was demonstrated on a reasonably large parking system on the NJIT campus in Newark, NJ.

In the future, the system is expected to be integrated to a Geographic Information System (GIS) in order to provide the users route planning information to the assigned parking lot, provide real-time information on the number of free parking spaces at each parking lot, parking costs for each parking lot and other pertinent information. In addition, the system will be designed in such a way such that any parking reservation algorithm can be easily implemented given that the corresponding solver can be easily integrated with the web-PRS interface.

ACKNOWLEDGEMENT

This project is sponsored by the New Jersey Commission on Science and Technology through the Transportation Information and Decision Engineering (TIDE) center at NJIT. Dr. John Tavantzis of the Mathmatics Department of NJIT helped in formulating the model. Mr. Elvis John helped in collecting data and part of programming.

REFERENCE

1. Mouskos, K. C., Bernstein, D., and Tavantzis, J. Mathematical Formulation of Deterministic and Stochastic Parking Reservation Systems (PRS) with Fixed Costs. Submitted for Publication to the *Transportation Research - Part C Journal*, 2002.
2. Fowler, M. and Scott, K. *UML Distilled: a brief guide to the standard object modeling language*. Addison-Wesley, 2000.
3. *Java Servlet API Specification Version 2.3*. September 2001, Sun Microsystems, Inc.
4. D. Ayers, H. Bergsten, M. Bogovich, J. Diamond, M. Ferris, M. Fleury, a. Halberstadt, P. Houle, P. Mohseni, A. Patzer, R. Philips, S. Li, K. Vedati, m. Wilcox and S. Zeiger. *Professional Java Server Programming.* Wrox Press Ltd., 2000.
5. XML Schema. http://www.w3.org/XML/Schema. Accessed July 25, 2002.
6. ILOG CPLEX. http://www.ilog.com/products/cplex/index.cfm. Accessed July 25, 2002.
7. ACME Java. http://www.acme.com/java/. Accessed on July 25, 2002.
8. T. Porter and T. Duff. *Compositing Digital Images*. Computer Graphics Volume 18, Number 3 July 1984 pp 253-259.
9. Blending Images.
http://developer.java.sun.com/developer/JDCTechTips/2002/tt0618.html#tip2. Accessed July 25, 2002.

LIST OF FIGURES

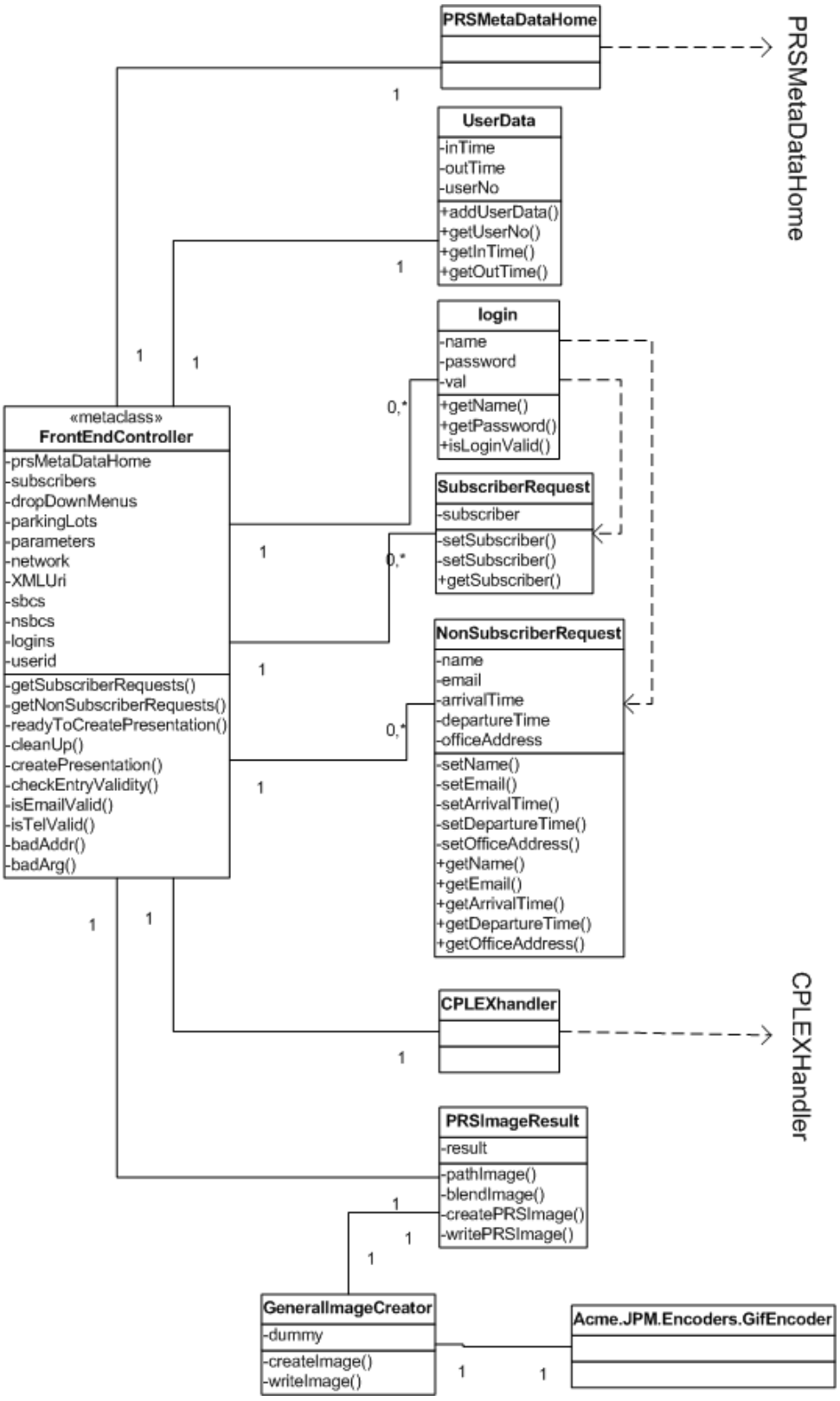

FIGURE 1 FrontEndController Class Diagram

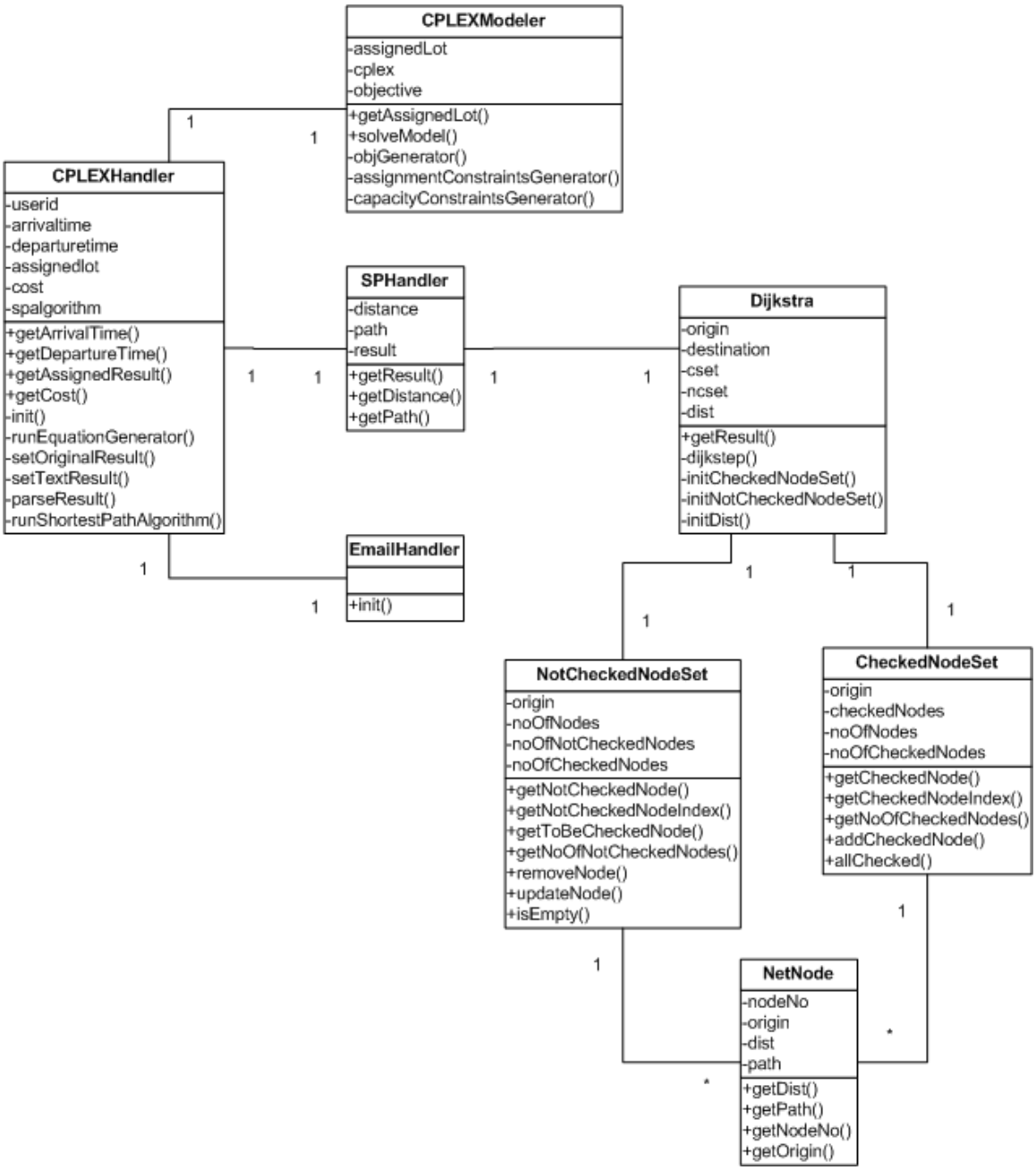

FIGURE 2 CPLEXHandler Class Diagram

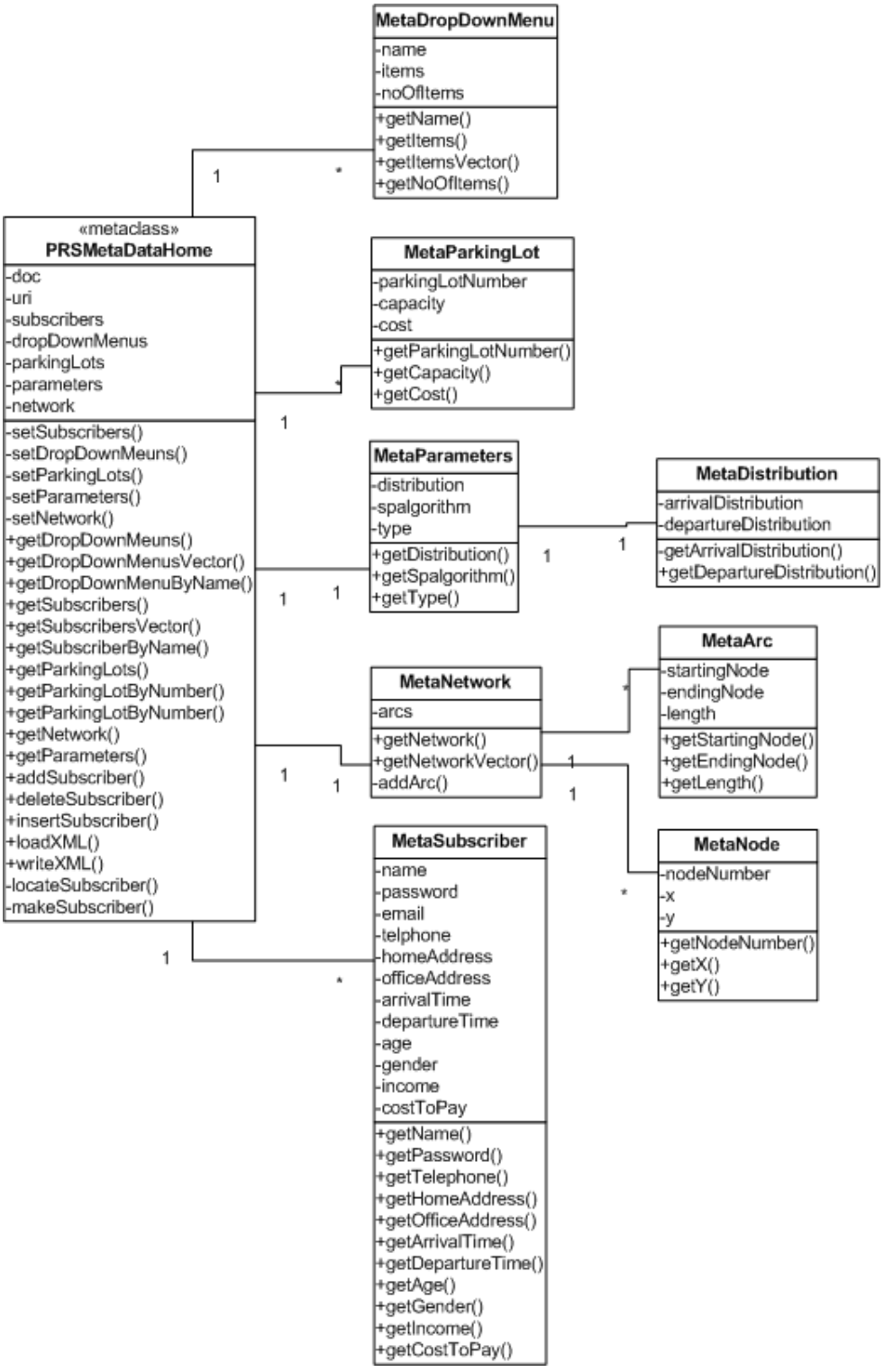

FIGURE 3 PRSMetaDataHome Class Diagram

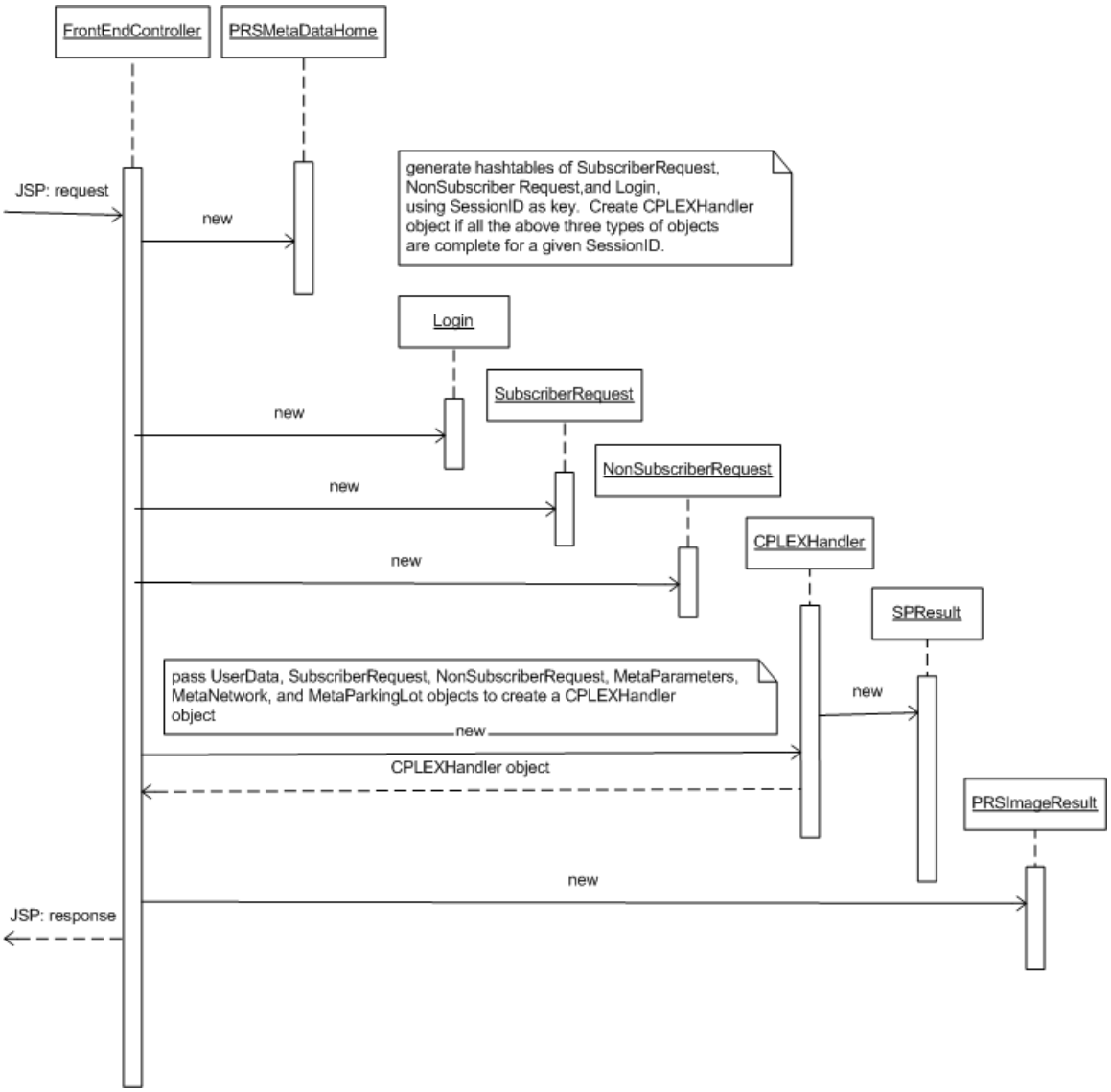

FIGURE 4 Web-PRS Sequence Diagram

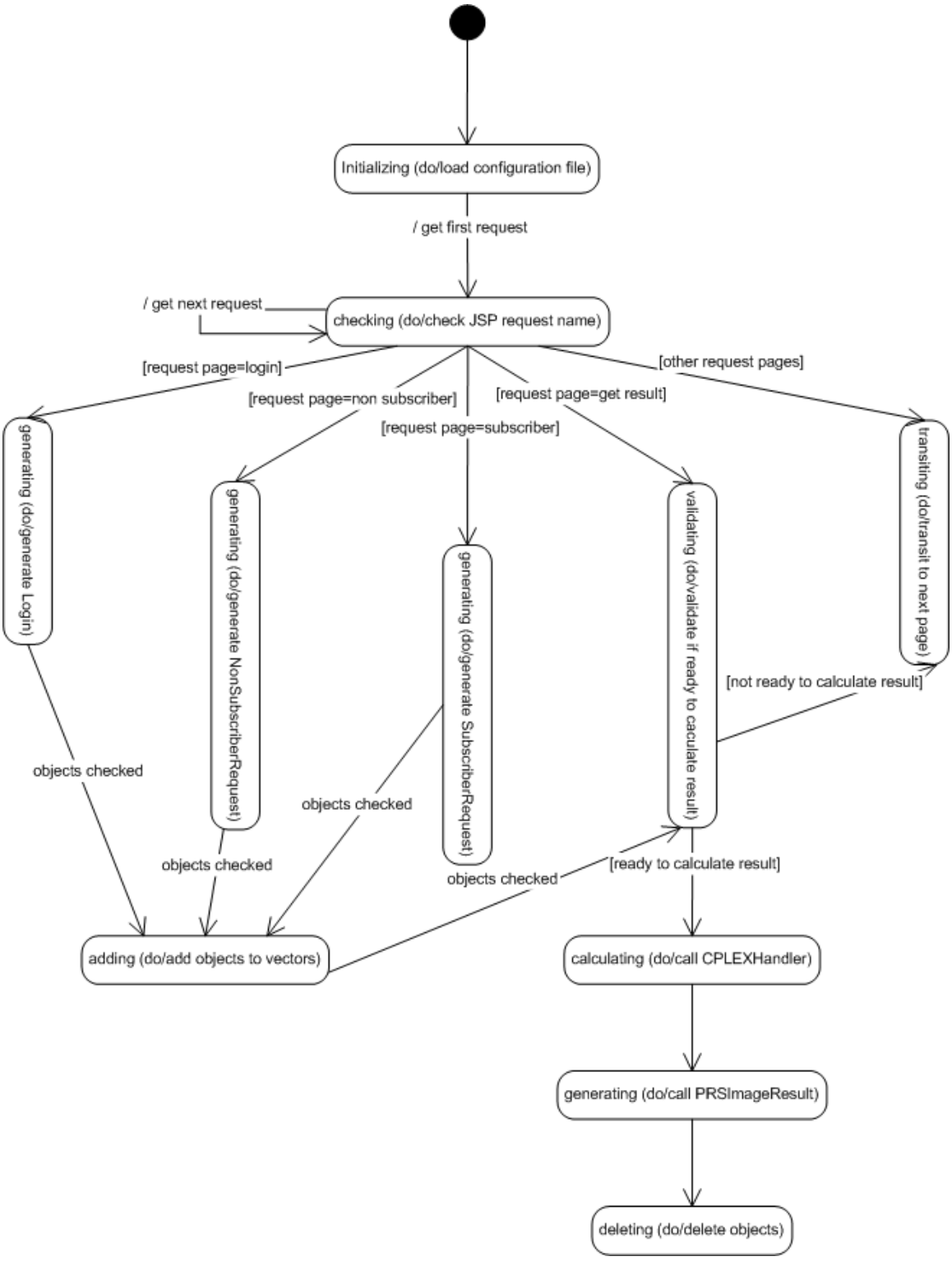

FIGURE 5 Web-PRS State Diagram

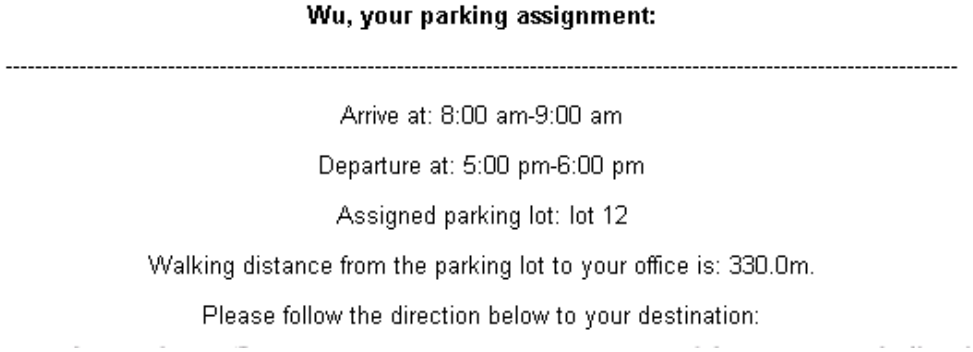

FIGURE 6 Parking Assignment Result-Text

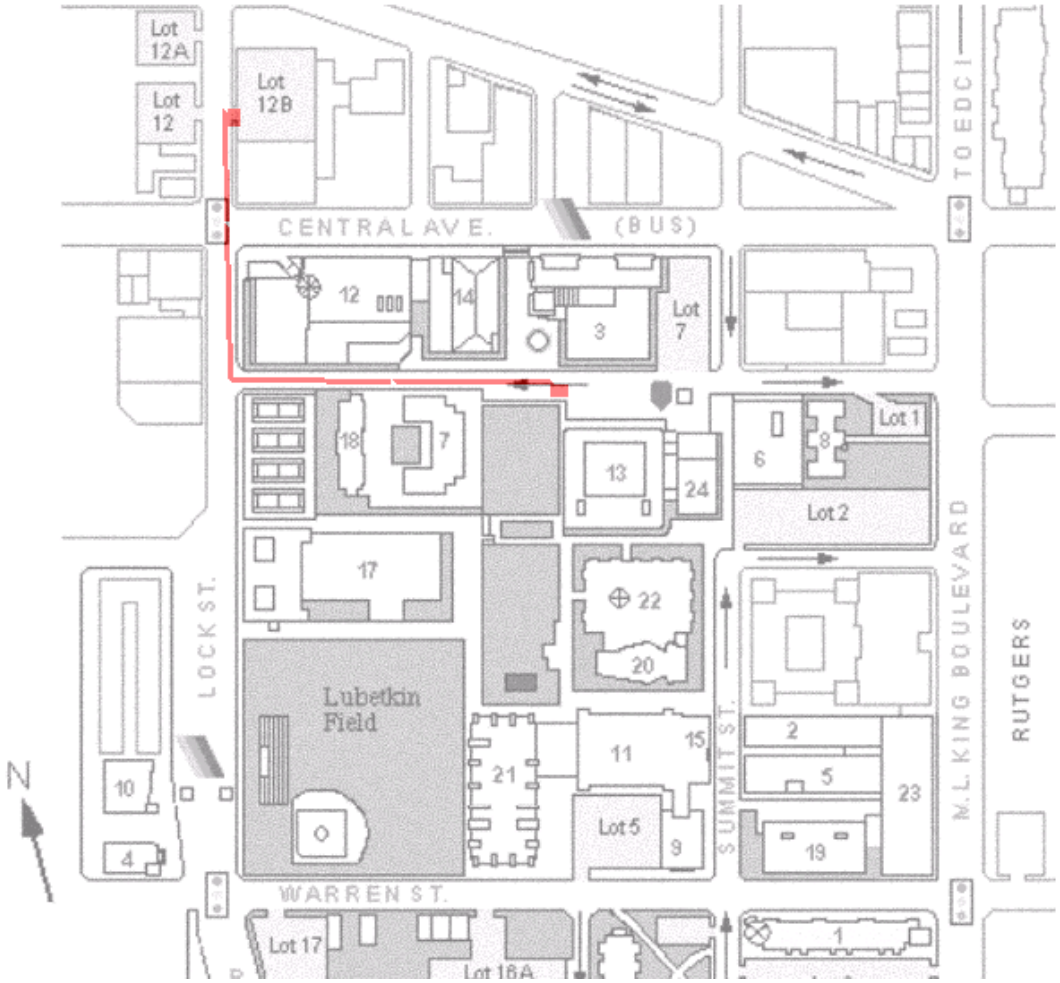

FIGURE 7 Parking Assignment Result-Graphic